高级程序语言实验报告

---数独游戏

装

订

线

姓名：童佳燕

学号：1551445

班级：计算机科学与技术1班

指导教师：沈坚

完成日期：2017.4.4

1.实验题目：数独游戏实现过程

1.1 问题描述（游戏规则）：

根据 9×9 盘面上的已知数字，在所有剩余空格处填入数字，并满足每一行、每一列、每一个粗线宫内的数字均含 1-9，不重复。

1.2 题目要求：

1.2.1 整体要求：

(1)所有小题放在一个程序中，以菜单的形式进行选择，包括手动+字符数组形式，手动+伪图形界面形式，自动+伪图形界面形式。

(2) 手动形式输入要求为（如 1b4，即在第 1 行 b 列，输入数字 4），要求记录每一步，并且可以一次回退

(3)要求提供文件选择功能，利用上下键进行选择，回车键读取文件信息，即题面即由用户自己选择数独谜面文件进行游戏。

1.2.2 显示要求：

（1）谜面提供的固定的数字和用户填写的数字要求不同显示

（2）自动求解形式要求求解过程以动画形式体现

（3）初始值以及游戏过程中，发生冲突的行，列以及宫要求不同的显示进行提醒

1.2.3 代码要求

（1）尽量使各菜单项中的程序公用函数，用参数解决细微差异。

（2）各函数代码长度尽量不超过 50 行

（3）不允许使用全局变量，全局指针以及全局数组，允许使用宏定义。

（4）只允许使用目前为止讲过的内容（包括课后补充知识。）

2.整体设计思路（以题目 3 为主）

第三题主要由三个部分组成：界面打印，文件选择，自动求解。

其中界面打印包括：固定值，待填充值的特殊显示；自动求解过程显示。

文件选择过程包括：文件目录的获取；筛选符合要求的文件；用户通过上下键滚动翻看文件回车键选择文件；程序读取文件内容。

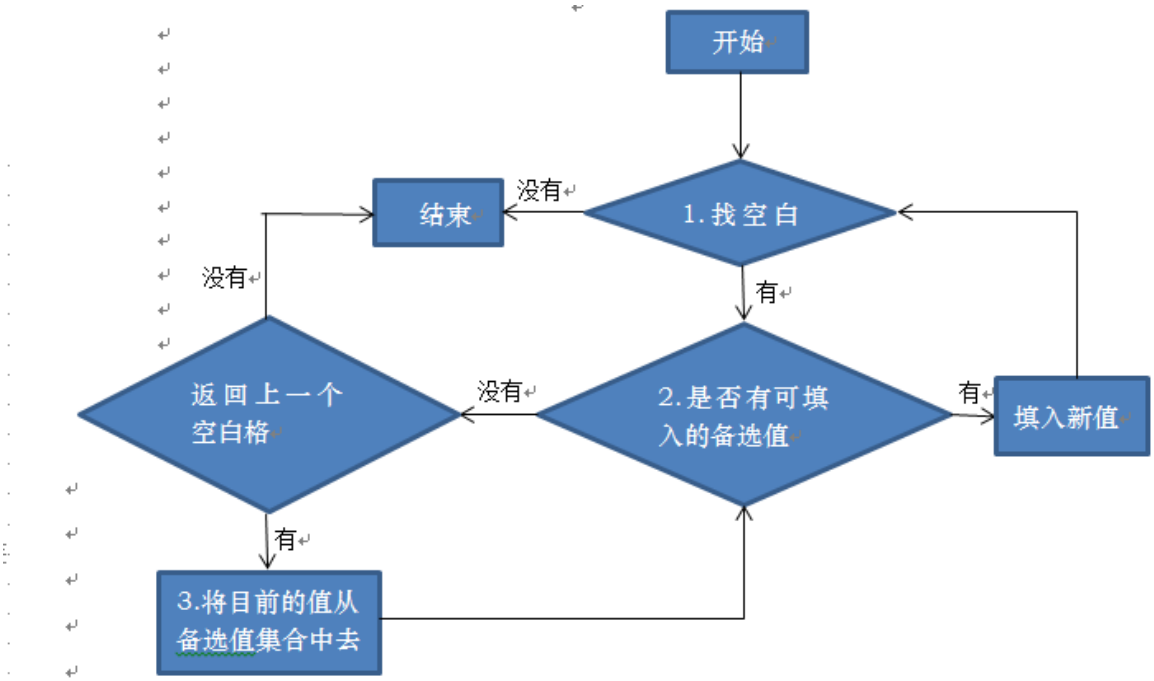
对于数独自动求解的算法，结合自己玩数独时使用的方法，技巧，主要考虑到有三种方式具体流程图如下：

装

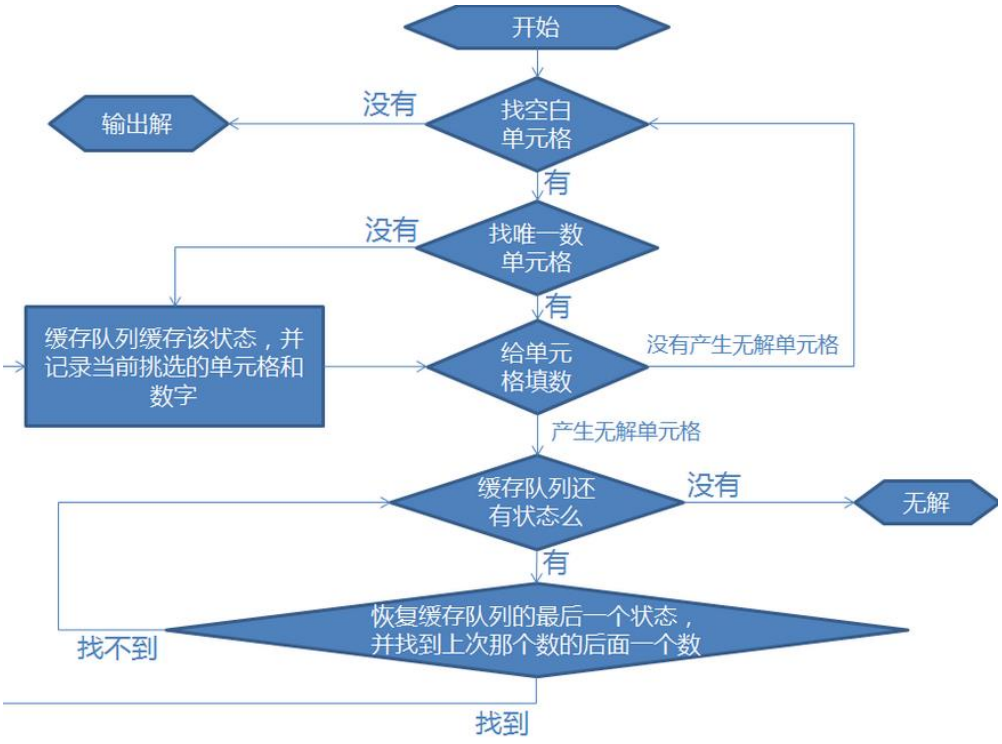
订

线

基础版：

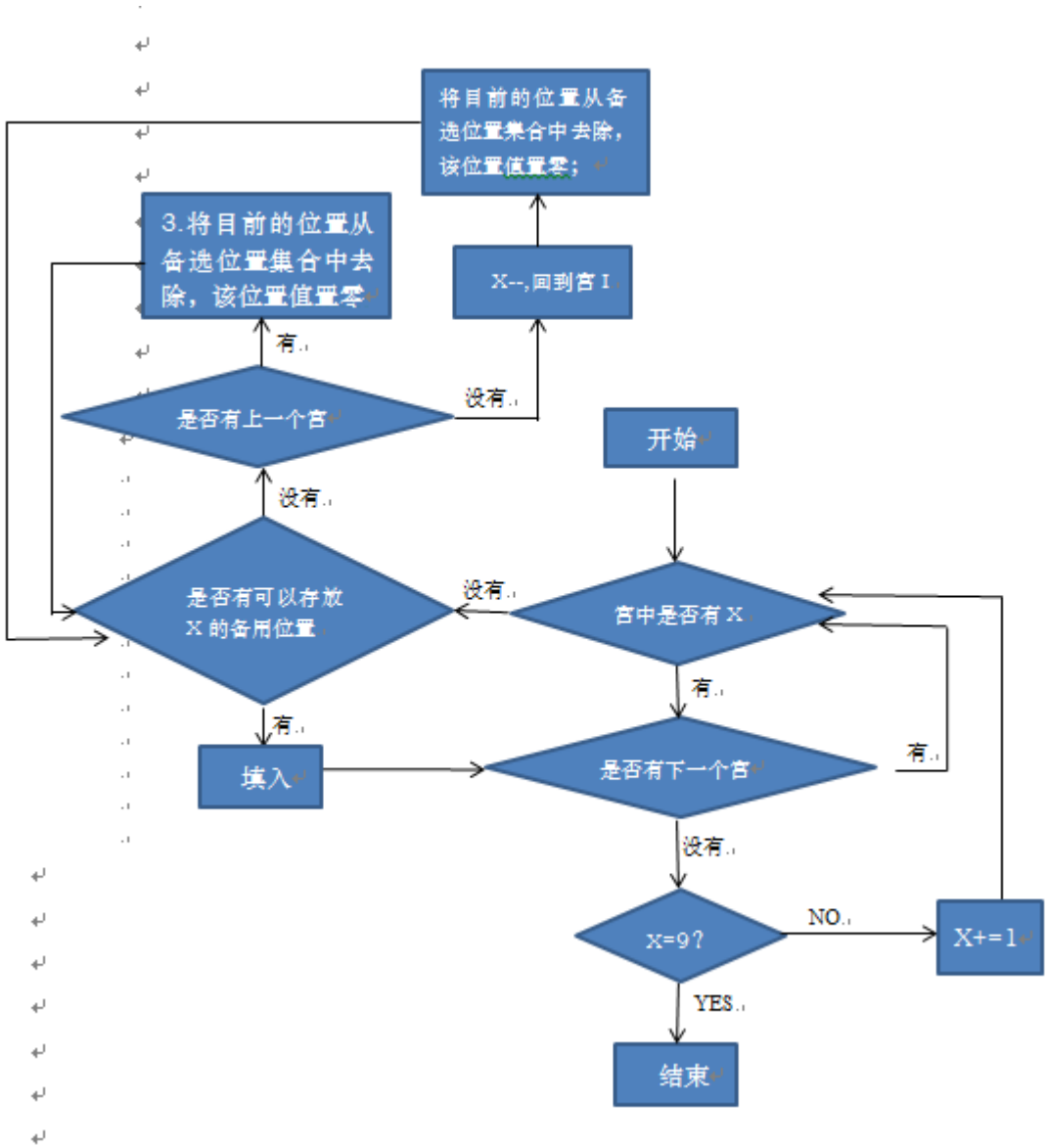

开始
1.找空白
没有
结束
有
2.是否有可填入的备选值
没有
返回上一个空白格
有
填入新值
没有
有
3.将目前的值从备选值集合中去

改良版：


开始
找空白单元格
没有
输出解
有
找唯一数单元格
没有
有
缓存队列缓存该状态，并记录当前挑选的单元格和数字
给单元格填数
没有产生无解单元格
产生无解单元格
缓存队列还有状态么
没有
无解
有
恢复缓存队列的最后一个状态，并找到上次那个数的后面一个数
找不到
找到

个人玩法版：

【以 3*3 宫为基本单位，分别标号 A-I 他，图中的 X 代表要填入的数字[1,9]】



由于时间原因以及编程的复杂难度，我选用了基础版的思路，利用回溯算法以及 DFS 相关算法进行求解，其中的利用数组存储单元格位置以及值的信息参考网上的方法，觉得很精妙，因此尝试着使用该方法。

该算法的效率确实是不够高的，不过我还是挺有兴趣对剩下两种算法进行编程，验证其效率；以及初次之外的其他算法，对该问题进行改良；

此处标记：未完待续。

3. 主要功能的实现

3.1 文件选择

3.1.1 读取符合规定的文件目录

该功能的实现是查的网上的资料，直接赋值粘贴了一段代码下来，大致自己理解了一下，进行了一定程度上的修改以适应要求。

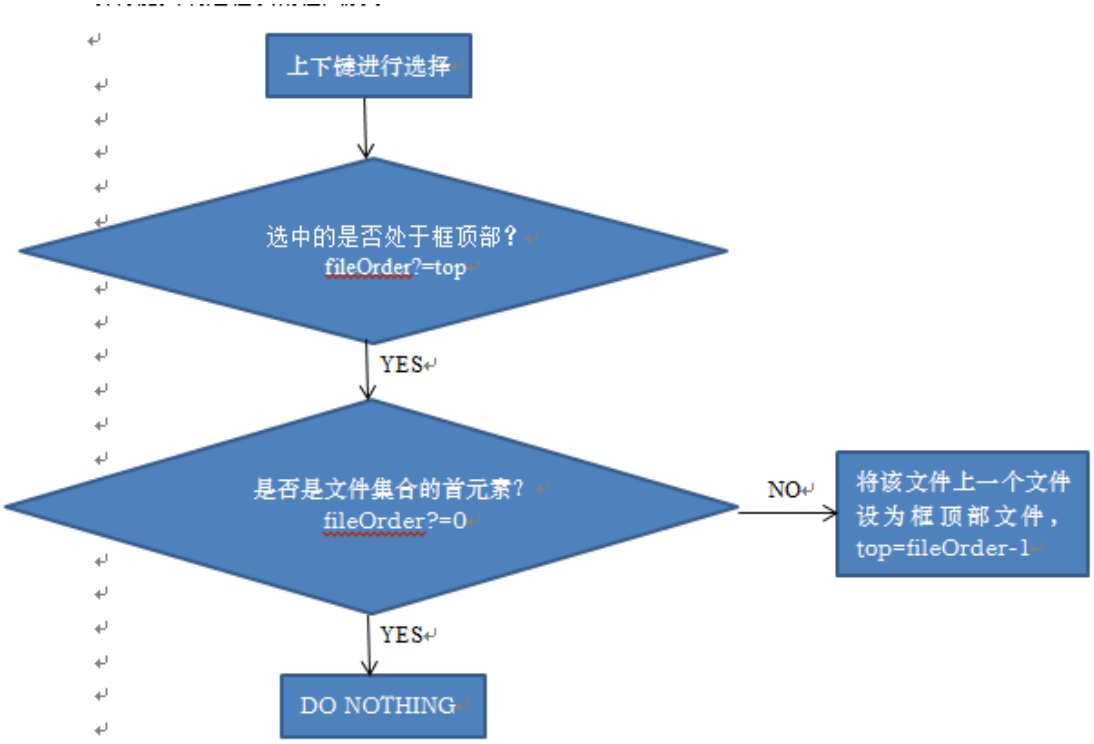
3.1.2 用户利用上下键翻看文件目录

该功能之一：用户按上下键，对应的文件名以不同于其他文件名的显示来提示被选中。

该功能本质上就是，利用上下键，“选中”对应文件名，对其进行特殊显色打印，同时对前一个（之前选中的一个）进行恢复正常显色操作；也就是需要在两处各打印一次。

功能之二：当选项框之“外”还有文件时，利用上下键可以进行“滚动”查看。

该功能实现过程以流程图展示：



本质上就是从下标为 top 到下标为 top+框的长度，也就是一个框中可以容纳的文件没那个的个数，这些文件的打印。

3.2 自动求解

3.2.1 空白格信息的存储

自动求解使用的算法是上述的基本算法，首先，由于遇到无备用值时需要有回溯操作，因此需要把所有的空白格的信息存储起来。本次作业中采用的是利用数组来进行存储。

主要的方法如下：rowStore[i][j]表示第 i 行有数字 j，返回类型是 Bool,也就是说如果第 i 行有数字 j，则返回 1；同样对于 colStore[i][j].而对于一个 3*3 的小宫格来说，blockStore[i][j][k]代表，以第 i 行第 j 列对应的单元格为左上角的 3*3 小宫格中有数字 k.返回值同样是 BOOL 型.

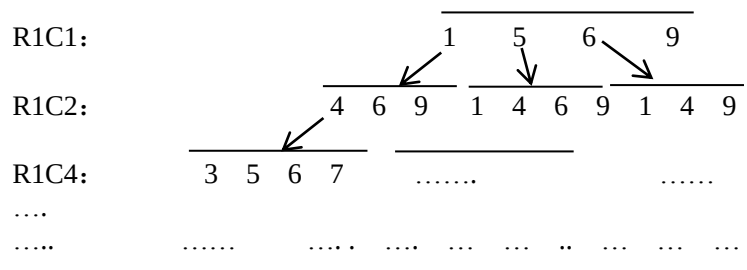
程序过程中，首先定义 colStore,rowStore,blockStore 的逻辑型数组，并且将其元素全都初始化为 false；然后对 sudoku 数组中的元素进行遍历，遇到非零值 sudoku[i][j]=num,则可以得到信息，rowStore[i][num]=colStore[j][num]=blockStore[i/3][j/3][num]=1;也就是将数独数组中冲突的判断存储在数组中。

该方法比单独写一个行列宫冲突的判断要方便，精妙，因此从网友手中“截取”过来，尝试变成自己的东西。

3.2.2 回溯算法+DFS 求解

回溯算法结合深度优先遍历方法以树的形式表示如下（以如下题面为例）：

	C1	C2	C3	C4	C5	C6	C7	C8	C9
R1			8				2		
R2		3		8		2		6	
R3	7				9				5
R4		5						1	
R5			4				6		
R6		2						7	
R7	4				8				6
R8		7		1		3		9	
R9			1				8		



一直到找出解或者所有路相通的顶点都被访问过而没有找到解，输出无解提示。

在实现过程中，利用一重循环对 1-9 所有值进行遍历，如果通过 rowStore,colStore,blockStore 进行判断后，符合不冲突的条件，则对 sudoku 中对应的值进行赋值，然后进入下一轮 DFS；如果遍历完所有的值都布满组条件，则 return;回到上一轮 dfs，并将对应的 bool 值进行标记，同时将对应的 sudoku 数组中值置零；直至找出解或者判断为无解为止。

4. 调试过程中遇到的问题

4.1 回溯算法有两个 return

调试过程中遇到一个问题就是明明已经找到解了，但是还是没有结束，查找次数远远大于应有的。

后来调试发现，算法的初衷是找到解就停止，也就是说，除了在遇到无备用值情况下回到上一步用到的 return ，还需要额外的 return 来解决当找到问题解时程序的退出。

4.2 未对回溯回来之后对应的值置零

调试过程中得不到正确的解，且在解决第一个问题之后，查找次数还是远远大于应有的在还没有得到正确的解之前，程序就停止了；或者说是，在找到正确解之后，程序还继续执行下去，却莫名其妙的停下。

4.3 依旧存在的问题

（1） 文件选择时按回车键默认 sudoku.txt 文件

（2）伪图形界面打印时，滚动键盘滚轮，界面就乱了；猜测是字体以及字体的大小发生的改版，尚未解决。

5. 心得体会

5.1 经验教训

（1） 这次程序全部使用的是驼峰命名法，用以规范自己的代码风格，与以往的下划线

命名法相比，在同行中相对更加主流，也为自己的代码能被别人看懂做一些改进。
（2）strcpy 的使用，首先，需要 cstring 头文件，其次，第一个参数必须是字符数组，而不能是“流放”的指针，最后，在 vs2015 中，需要使用 strcpy_s
（3）写程序的流程还是要规范化一些，事先设计整体流程，理出主要的步骤，功能；设计函数，参数，命名等。

事先设计好之后，在动工写代码，可以一定程度上减少比如说，写到一半发现，类似的功能，需要写两个完全不同的函数去解决不同情况下问题的解决。

就像此次代码中，判断是否有行列宫冲突的时候，使用的是两种完全不同的思路。主要是因为在写第一个函数的时候，没有考虑到自动求解是对于“摒弃备用值”的需求。

由于时间原因，也没有对其进行统一，导致代码显得复杂，冗余。

5.2 问题分解与函数重用

5.2.1 问题分解

在正式开始写程序以及老师下发具体要求之前，写过设计报告！在老师问题的引导下，基本确定了整个程序的基本思路，包括内部以什么存储方式，怎么寻路，怎么判断连线得分，将几个主要的问题想清楚之后，剩下的就是往里面加装饰，以及具体的操作过程中遇到大大小小的错误。

将一个大的问题拆分成一个个小的问题，然后再解决每个小问题的时候，再用不同的函数去解决更小的问题。写程序某种程序上来说，就是一种逐步递进的过程。

最后的组合，我相信，熟能生巧，不是一天两天能练成的，但是平时多注意，会掌握的。

5.2.2 函数重用

在写程序之前，先看了题目要求，前三题都是打印内部数组，后三题需要打印伪图形界面。显然，一个内部数组的打印需要被用到很多次，而且每次要求都是不同的。为了方便，同时为了减少重负冗杂代码，随着函数功能的增加，逐渐增加参数来满足需求。

其实我在写程序，整理思路的时候，基本就可以把要写哪几个函数，对应要把哪些问题解决，大致可以确定下来，当然在实践过程中碰到新的问题继续改进，但基本大的函数应该不会变。只有尽可能的去满足功能的需求。

这次作业做得比较毛糙，尤其是最后两个小题，没有进行合理的函数拆分和组合，一股脑儿全堆在一起,但我想，考试过去有时间我一定会重新整理完善的！

6. 源代码(主要函数)

```
int gameAuto()
{
    const HANDLE hout = GetStdHandle(STD_OUTPUT_HANDLE);
    setconsoleborder(hout, 100, 40, 40);//内含清屏

    int sumOfFile = 0, isInvalid = 0, x, y;
    char fileName[30];
    struct Sudoku sudoku[N][N];
    Place points[81];

    char(*file)[30] = (char(*)[30])malloc(20 * 30 * sizeof(char));
    string path = "E:\\程设作业\\数独\\数独";
    dir(path, file, &sumOfFile);
    fileMenu(file, sumOfFile);
    int fileOrder = chooseAction(file, sumOfFile);
    strcpy_s(fileName, file[fileOrder]);

    //bug,拖动滚动条，显示会出现变动!
    if (getValueFromFile(sudoku, fileName, 1) == -1)
        return -1;
    if (isRight(sudoku) == -1)
    {
        cout << "文件数据错误" << endl;
        return -1;
    }
    init(sudoku);
    if (printConsoleAllFrame(sudoku, &isInvalid))
    {
        cout << "数据有冲突" << endl;
        return -1;
    }
    getxy(hout, x, y);

    int rowStore[N][N], colStore[N][N], blockStore[3][3][N];
    for(int i=0;i<N;i++)
        for (int j = 1; j <=N; j++)
        {
            rowStore[i][j] = 0;
            colStore[i][j] = 0;
        }
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            for (int k = 1; k <= N; k++)
                blockStore[i][j][k] = 0;

    int num = allocateEmptyPoints(points, sudoku,rowStore,colStore,blockStore);
    int flag = 0,paceNum = 0;
    DFS(num, points, sudoku,&flag,rowStore,colStore,blockStore,&paceNum);
    cout << endl;
    return 0;
}
int allocateEmptyPoints(Place points[81], Sudoku(*sudoku)[N],int rowStore[][N],int colStore[][N],int
blockStore[][3][N])
{
    int num = 0;
```

```
    for(int i=0;i<N;i++)
        for (int j = 0; j < N; j++)
        {
            if (sudoku[i][j].value == 0)
            {
                points[num].x = j;
                points[num++].y = i;
            }
            else
                rowStore[i][sudoku[i][j].value] = colStore[j][sudoku[i][j].value] =
blockStore[i/3][j/3][sudoku[i][j].value] = 1;

        }
    return num - 1;
}
void DFS(int n,Place points[81], Sudoku(*sudoku)[N],int *flag, int rowStore[][N], int colStore[][N],
int blockStore[][3][N],int *paceNum)
{
    const HANDLE hout = GetStdHandle(STD_OUTPUT_HANDLE);
    if (*flag)
        return;

    if (n == -1)
    {
        *flag = 1;
        return;
    }

    (*paceNum)++;
    gotoxy(hout, 0, 30);
    cout << (*paceNum);

    for (int i = 1; i < 10 && !(*flag);i++)
    {
        int x = points[n].x;
        int y = points[n].y;
        if (!rowStore[y][i] && !colStore[x][i] && !blockStore[y / 3][x / 3][i])
        {
            rowStore[y][i] = colStore[x][i] = blockStore[y/ 3][x / 3][i] = true;
            sudoku[y][x].value = i;
            showSearchAction(x, y, sudoku);//不同颜色打印填入值的单元格显示动画
            DFS(n - 1,points,sudoku,flag,rowStore,colStore,blockStore,paceNum);
            rowStore[y][i] = colStore[x][i] = blockStore[y / 3][x / 3][i] = false;
            sudoku[y][x].value = 0;
        }
    }
}
void dir(string path,char file[][30],int *sum)
{
    long hFile = 0;
    struct _finddata_t fileInfo;
    string pathName, exdName;
    if ((hFile = _findfirst(pathName.assign(path).append("\\*.txt").c_str(), &fileInfo)) == -1)
        return;
    strcpy_s(file[*sum], fileInfo.name);
    while (_findnext(hFile, &fileInfo) == 0)
```

```
    {
        (*sum)++;
        strcpy_s(file[*sum], fileInfo.name);
    }
    _findclose(hFile);
    return;
}

void fileMenu(char (*file)[30],int numOfFile)
{
    const int baseX = 60, baseY = 2,height=9,weight=23;
    const HANDLE hout = GetStdHandle(STD_OUTPUT_HANDLE);
    gotoxy(hout,baseX,baseY);
    cout << "数独样本文件";
    gotoxy(hout, baseX, baseY + 1);
    cout << "┌──────────────────────┐";
    for (int i = 0; i < height; i++)
    {
        gotoxy(hout, baseX, baseY + 2 + i);
        cout << "│ ";
        if (i == 0)
            setcolor(hout, COLOR_WHITE, COLOR_BLACK);
        cout << file[i];
        setcolor(hout, COLOR_BLACK, COLOR_WHITE);
        gotoxy(hout, baseX + weight-1, baseY + 2 + i);
        cout << "│";
    }
    gotoxy(hout, baseX, baseY +height+1);
    cout << "└──────────────────────┘\n";

}

int chooseAction(char(*file)[30],int numOfFile)//改进的时候，尝试拆分
{
    const HANDLE hout = GetStdHandle(STD_OUTPUT_HANDLE);
    int baseX = 60, baseY = 2;
    int key1, key2;
    int fileOrder = 0, top = 0;

    while (1)
    {
        key1 = _getch();
        if (key1 == '\r')
            break;
        key2 = _getch();
        if (key1 == 224 && key2 == 80)//向下键
        {
            if (fileOrder == numOfFile - 1)
                continue;//以指向最后一个文件，该次输入不生效
            fileOrder++;
            if (fileOrder == top + 8)//所指向的file
            {
                top++;
                for (int i = 0; i < 7; i++)
                {
                    gotoxy(hout, baseX + 3, baseY + 2 + i);
```

装 订 线

```
                    cout <<"                    ";
                    gotoxy(hout, baseX +3, baseY + 2 + i);
                    cout << file[top + i];
                }
                gotoxy(hout, baseX + 3, baseY + 9);
                cout << "                    ";//清空该位置之前打印的信息
                gotoxy(hout, baseX + 3, baseY + 9);
                setcolor(hout, COLOR_WHITE, COLOR_BLACK);//设置颜色
                cout << file[top + 7];
                setcolor(hout, COLOR_BLACK, COLOR_WHITE);
            }
            else
            {
                gotoxy(hout, baseX + 3, baseY + 2 + fileOrder - top-1);
                setcolor(hout, COLOR_BLACK, COLOR_WHITE);
                cout << file[fileOrder-1];
                gotoxy(hout, baseX + 3, baseY + 2 + fileOrder - top);
                setcolor(hout, COLOR_WHITE, COLOR_BLACK);
                cout << file[fileOrder];
                setcolor(hout, COLOR_BLACK, COLOR_WHITE);
            }
        }
        else if (key1 == 224 && key2 == 72)
        {
            if (fileOrder == 0)
                continue;
            fileOrder--;
            if (fileOrder == top - 1)
            {
                top--;
                gotoxy(hout, baseX + 3, baseY + 2);
                cout << "                    ";
                gotoxy(hout, baseX + 3, baseY + 2);
                setcolor(hout, COLOR_WHITE, COLOR_BLACK);
                cout << file[top];
                setcolor(hout, COLOR_BLACK, COLOR_WHITE);

                for (int i = 1; i <= 7; i++)
                {
                    gotoxy(hout, baseX + 3, baseY + 2 + i);
                    cout << "                    ";
                    gotoxy(hout, baseX +3, baseY + 2 + i);
                    cout << file[top + i];
                }
            }
            else
            {
                gotoxy(hout, baseX + 3, baseY + 2 + fileOrder - top+1);
                setcolor(hout, COLOR_BLACK, COLOR_WHITE);
                cout << file[fileOrder+1];
                gotoxy(hout, baseX + 3, baseY + 2 + fileOrder - top );
                setcolor(hout, COLOR_WHITE, COLOR_BLACK);
                cout << file[fileOrder];
                setcolor(hout, COLOR_BLACK, COLOR_WHITE);
            }
        }
    }
```

装 订 线

```
    gotoxy(hout, 0, 0);
    return fileOrder;
}
```

装

订

线